ТЕХНИКИ (ТАКТИКИ) ЗА ПОСТИГАНЕ НА КАЧЕСТВОТО НА СОФТУЕРА

Част 1

Преговор

- В досегашните лекции разгледахме
  - Най-често изискваните от една софтуерна система *качества* (нефункционални изисквания)
  - Как те се формализират, така че дефинициите им да не се припокриват и разделихме качествата на системни (технологични), бизнес и архитектурни.
  - Няколко подходящи архитектурни решения, доказали се в практиката, които решават конкретни проблеми
- В настоящата лекция ще обобщим шаблоните като разгледаме на абстрактно ниво конкретни техники (тактики) за удовлетворяване на качествените изисквания.

Въведение

- Как става така, че един дизайн притежава висока надеждност, друг висока производителност, а трети – висока сигурност?
- Постигането на тези качества е въпрос на фундаментални архитектурни решения – тактики.
- Тактиката е архитектурно решение, чрез което се контролира ***резултата*** на даден сценарий за качество.
- Наборът от конкретни тактики се нарича архитектурна стратегия.

На днешната лекция

- Тактики за постигане на
  - Изправност/наличност (dependability/availability)
  - Производителност (performance)

Тактики за изправност

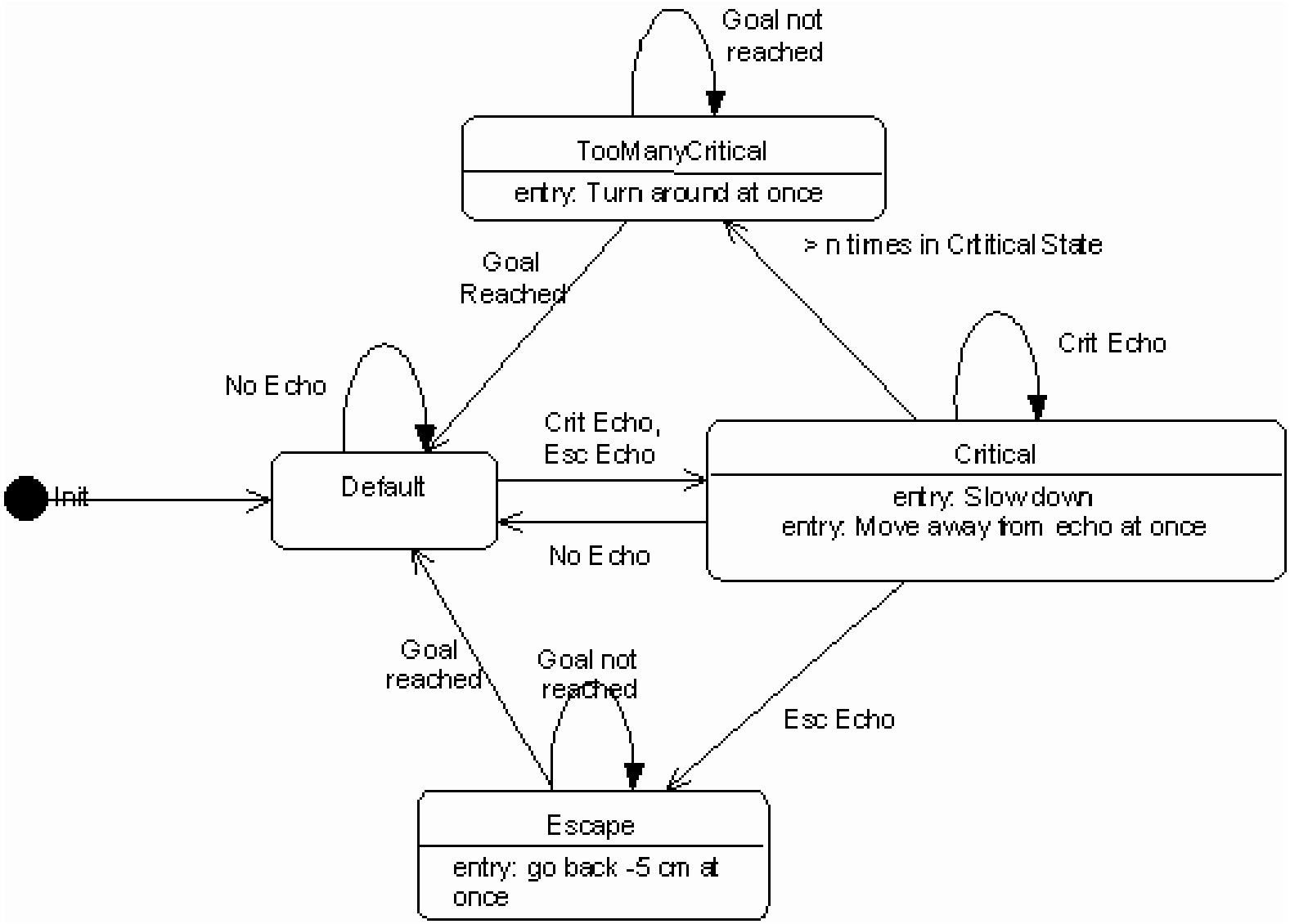
- Откриване и предпазване от откази
- Отстраняване на откази
- Повторно въвеждане в употреба

ТАКТИКИ ЗА ИЗПРАВНОСТ – ОТКРИВАНЕ НА ОТКАЗИ

Откриване на откази

- ***Exo (Ping/echo)*** – компонент А пуска сигнал до компонент Б и очаква да получи отговор в рамките на определен интервал от време. Ако отговорът не се получи навреме, се предполага, че в компонент Б (или в комуникационния канал до там) е настъпила повреда и се задейства процедурата за отстраняване на повредата.
- Използва се напр. от група компоненти, които солидарно отговарят за една и съща задача; от клиенти, които проверяват дали даден сървърен обект и комуникационния канал до него работят съгласно очакванията за производителност.
- Вместо един детектор да ping-ва всички процеси, може да се организира йерархия от детектори – детекторите от най-ниско ниво ping-ват процесите, с които работят заедно върху един процесор, докато детекторите от по-високо ниво ping-ват детекторите от по-ниско ниво и т.н. – така се спестява мрежови трафик;

- В какви структури (изгледи) е подходящо да се опише тактиката за Exo?


Goal not reached
TooManyCritical
entry: Turn around at once
> n times in Critical State
Goal Reached
Crit Echo
No Echo
Crit Echo, Esc Echo
Init
Default
Critical
entry: Slow down
entry: Move away from echo at once
No Echo
Goal reached
Goal not reached
Esc Echo
Escape
entry: go back -5 cm at once

Откриване на откази

- ***Heartbeat, Keepalive*** – даден компонент периодично излъчва сигнал, който друг компонент очаква. Ако сигналът не се получи, се предполага, че в компонент А е настъпила повреда и се задейства процедура за отстраняване на повредата.
- Сигналът може да носи и полезни данни – напр., банкоматът може да изпраща журнала от последната транзакция на даден сървър. Сигналът не само действа като heartbeat, но и служи за лог-ване на извършените транзакции;

Откриване на откази

- ***Изключения (Exceptions)*** – обработват се изключения, които се генерират, когато се стигне до определено състояние на отказ.
- Обикновено процедурата за обработка на изключения се намира в процеса, който генерира самото изключение.

ТАКТИКИ ЗА ИЗПРАВНОСТ – ОСТСТРАНЯВАНЕ НА ОТКАЗИ

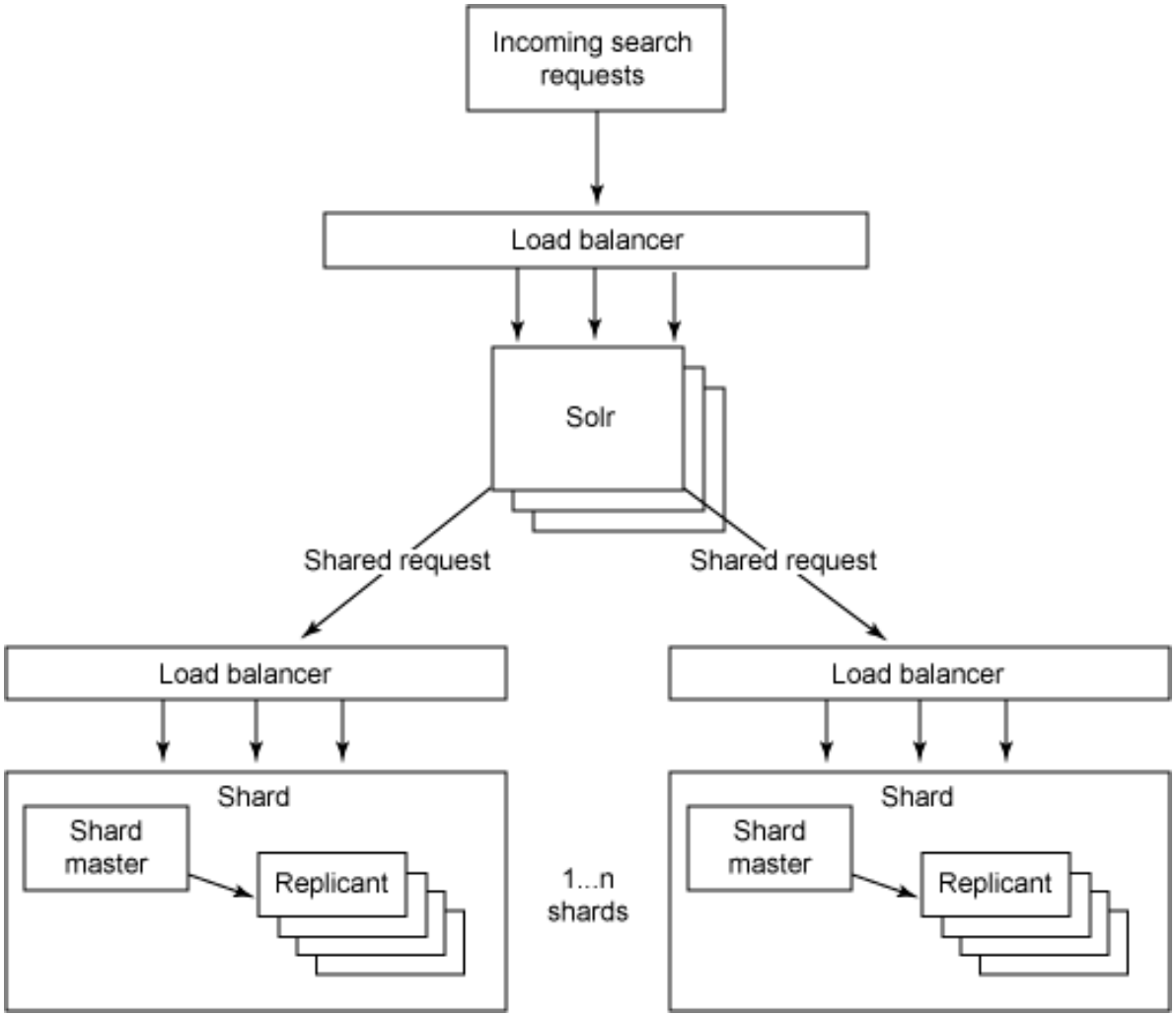
Отстраняване на откази

- ***Активен излишък (Active redundancy, hot restart)*** – важните компоненти в системата са дублирани (вкл. многократно). Дублираните компоненти се поддържат в едно и също състояние (чрез подходяща синхронизация, ако се налага това). Използва се резултатът само от единния от компонентите (т.н. активен);
- Обикновено се използва в клиент/сървър конфигурация, като напр. СУБД, където се налага бърз отговор дори при срив.
- Освен излишък в изчислителните звена се практикува и излишък в комуникациите, в поддържащият хардуер и т.н.
- Downtime-ът обикновено се свежда до няколко милисекунди, тъй като резервният компонент е готов за действие и единственото, което трябва да се направи е той да се направи активен.

Отстраняване на откази

- ***Пасивен излишък (Passive redundancy, warm restart)*** – Един от компонентите (основният) реагира на събитията и информира останалите (резервните) за промяната на състоянието.
- При откриване на отказ, преди да се направи превключването на активния компонент, системата трябва да се увери, че новият активен компонент е в достатъчно осъвременено състояние.
- Обикновено се практикува периодично форсиране на превключването с цел повишаване на надеждността
- Обикновено downtime-ът е от няколко секунди до няколко часа.
- Синхронизацията се реализира от активния компонент.

Пример – Apache solr



Отстраняване на откази

- ***Резерва (Spare)*** – поддръжка на резервни изчислителни мощности, които трябва да се инициализират и пуснат в действие при отказ на някой от компонентите.
- За целта може да е необходима постоянна памет, в която се записва състоянието на системата и която може да се използва от резервната система за възстановяване на състоянието.
- Обикновено се използва за хардуерни компоненти и работни станции.
- Downtime-ът обикновено е от няколко минути до няколко часа.

Основни предизвикателства

- Синхронизация на състоянието на отделните дублирани модули
- Данните трябва да са консистентни във всеки един момент
- Има ли *отстраняване на откази* при копиране на един и същи код?

Разнородност

- отказите в софтуера обикновено се предизвикват от грешки при проектирането
- Мултиплицирането на грешка в проектирането чрез репликация не е добра идея
- Просто увеличаване на броя идентични копия на програмата (подобно на техниките в хардуера) не е решение
- Трябва да се въведе разнородност в копията на програмата

Аспекти на разнородността в софтуерните системи

- Разнородност в проектирането (design diversity)
- Разнородност по данни (data diversity)
- Разнородност по време (temporal diversity)

Разнородност в проектирането

- Различни програмни езици
- Различни компилатори
- Различни алгоритми
- Ограничен/липса на контакт между отделните екипи
- Модул за избор на резултат от изпълнението на отделните копия

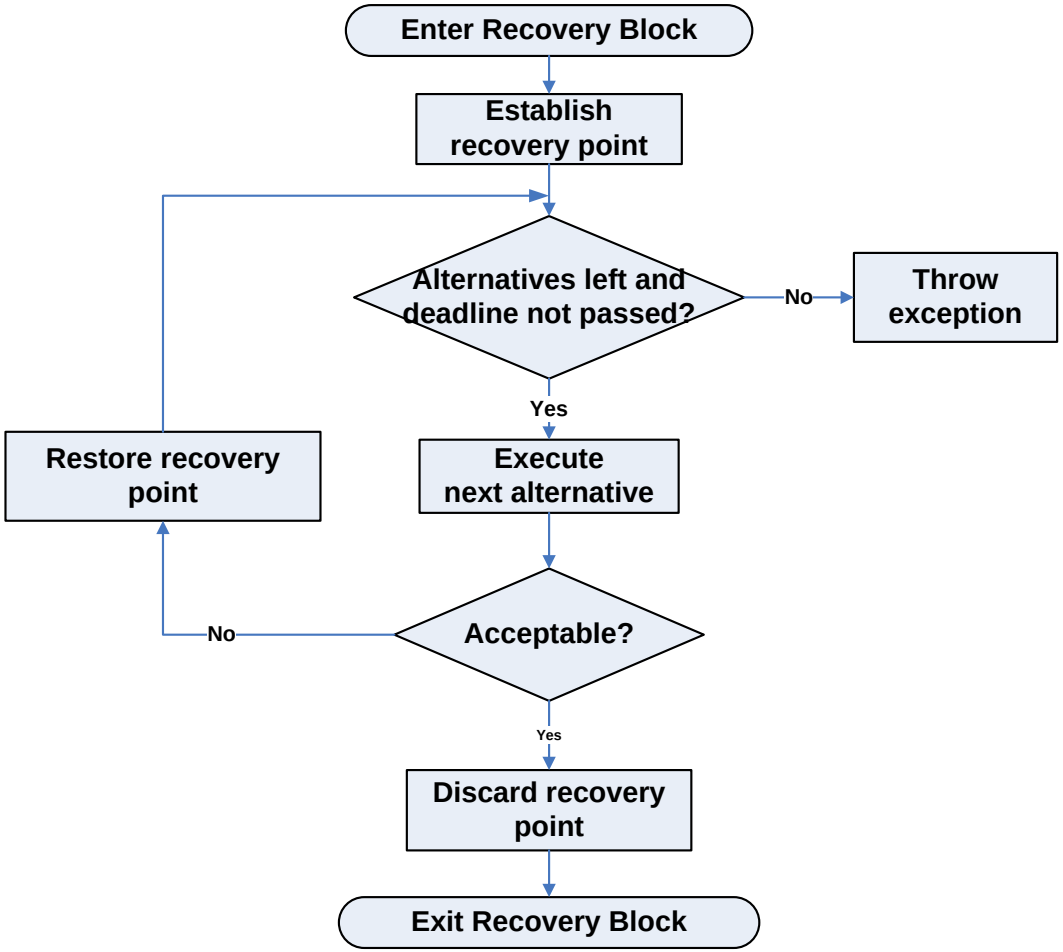
Техники за постигане на разнородност в проектирането

- Recovery Blocks
- Програмиране на N на брой версии (N-version programming)
- И др.

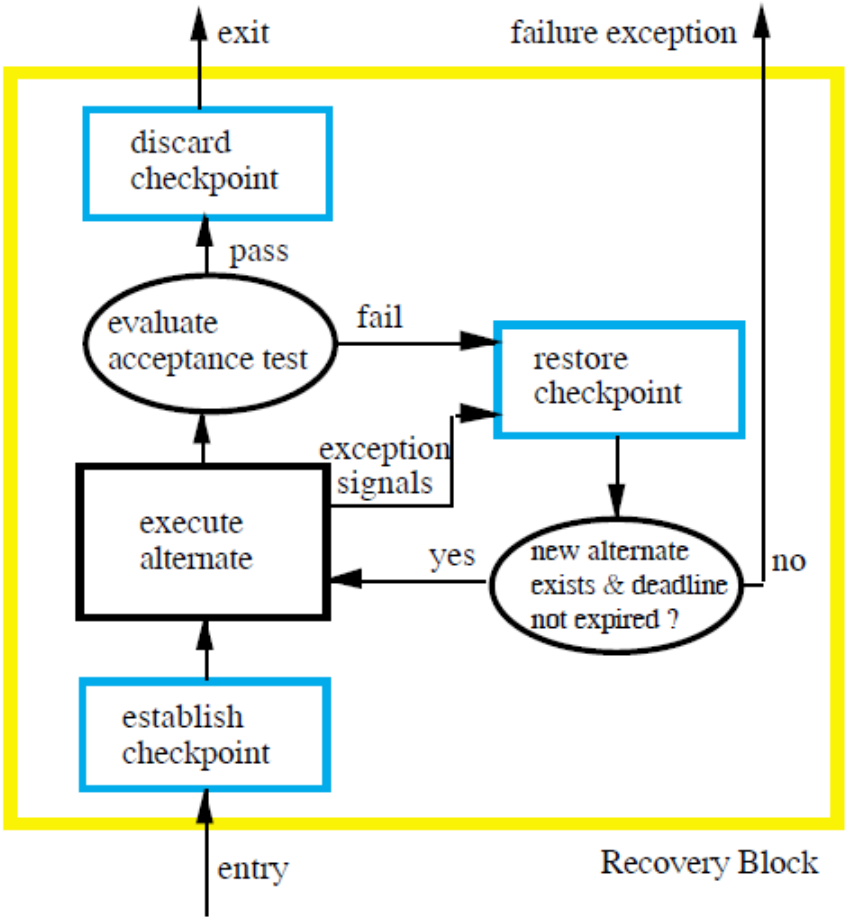
Recovery Blocks

- Разработват се няколко алтернативни модула на програмата
- Извършват се тестове за одобрение, за да се определи дали получения резултат е приемлив

Алгоритъм за recovery blocks

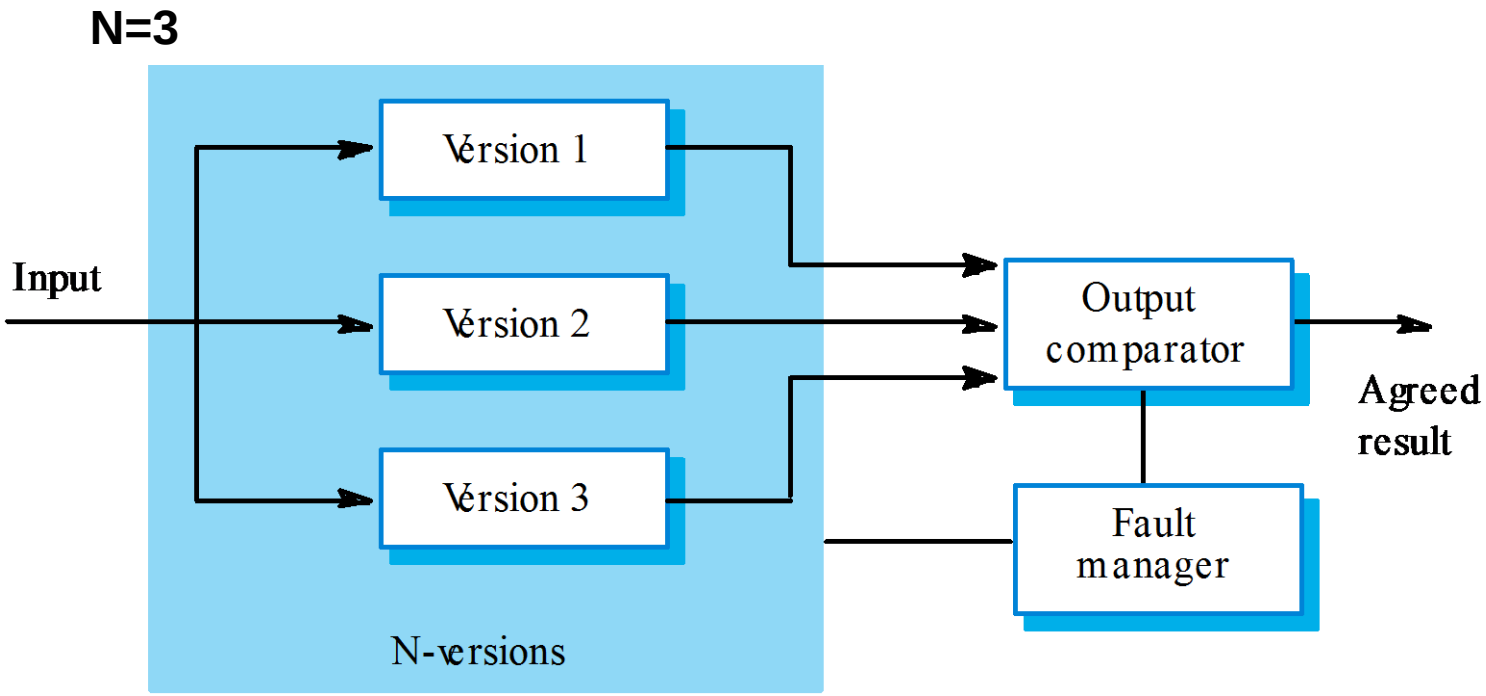


Принцип на Recoverv blocks



Източник: Randell, Brian, and Jie Xu. "The evolution of the recovery block concept.“ , 1995.

N-version programming



Източник: Sommervile, I. “Software Engineering”

N-version programming

- ***Voting*** - На различни процесори работят еквивалентни процеси, като всички те получават един и същ вход и генерират един и същ резултат, който се изпраща на т.н. voter (output comparator), който решава крайния резултат от изчислението. Ако някои от процесите произведе изход, който се различава значително от останалите, voter-ът решава да го изключи от обработката.
- Алгоритъмът за изключване на процес от обработката може да бъде различен и изменян, напр. отхвърляне чрез мнозинство, предпочитан резултат и т.н.

Пример за програмиране на N на брой версии

- Система за контрол на полета в самолетите Boeing 777
  - Използване е един език за програмиране - ADA
  - 3 различни среди за разработка
  - 3 различни компилатора
  - 3 различни процесора

Разнородност по време

- Предполага възникването на определени събития, които касаят работата на програмата, по различно време
- Методи за реализация на разнородност по време
  - Чрез стартиране на изпълнението в различни моменти от време
  - Чрез подаване на данни, които се използват или четат в различни моменти от времето

Тактики за Изправност – отстраняване на откази

- ***Извеждане от употреба (Removal from service)*** – премахва се даден компонент от системата, за да се избегнат очаквани сривове.
- Типичен пример – периодичен reboot на сървърите за да не се получават memory leaks и така да се стигне до срив.
- Извеждането от употреба може да става автоматично и ръчно, като и в двата случая това следва да е предвидено в системата на ниво архитектура.

Тактики за Изправност – отстраняване на откази

- ***Следене на процесите (Process Monitoring)*** – посредством специален процес се следят основните процеси в системата. Ако даден процес откаже, мониторинг процеса може да го премахне, пре-инициализира, да създаде нов екземпляр и т.н.

ТАКТИКИ ЗА ИЗПРАВНОСТ – ПРИ ПОВТОРНО ВЪВЕЖДАНЕ В УПОТРЕБА

Повторно въвеждане в употреба

- ***Паралелна работа (shadow mode)*** – преди да се въведе в употреба компонент, който е бил повреден, известно време се оставя той да работи в паралел в системата, за да се уверим, че се държи коректно, точно както работещите компоненти.

Тактики за Изправност – повторно въвеждане в употреба

- ***Ре-синхронизация на състоянието (State resynchronization)*** – тактиките за пасивен и активен излишък изискват състоянието на компонентите, които се въвеждат повторно в употреба да бъде ре-синхронизирано със състоянието на останалите работещи компоненти.
- Начинът, по който ще се извърши ре-синхронизацията зависи от downtime-а, който може системата да си позволи, размера на данните за ре-синхронизация и т.н.

Тактики за Изправност – повторно въвеждане в употреба

- ***Контролни точки и rollback (Checkpoint/rollback)*** – Контролната точка е запис на консистентно състояние, създаван периодично или в резултат на определени събития.
- Понякога системата се разваля по необичаен начин и изпада в не-консистентно състояние. В тези случаи, системата се възстановява (rollback) в последното консистентно състояние (съгласно последната контролна точка) и журнала на транзакциите, които са се случили след това.

ТАКТИКИ ЗА ПРОИЗВОДИТЕЛНОСТ

Тактики за производителност

- Целта на тактиките за производителност е да се постигне реакция от страна на системата на зададено събитие в рамките на определени времеви изисквания.
- За да реагира системата е нужно време, защото:
  - Ресурсите, заети в обработката го консумират;
  - Защото работата на системата е блокирана поради съревнование за ресурсите, не-наличието на такива, или поради изчакване на друго изчисление;
- Тактиките за производителност са разделени в три групи:
  - Намаляване на изискванията;
  - Управление на ресурсите;
  - Арбитраж на ресурсите;

Тактики за производителност – намаляване на изискванията

- Увеличаване на производителността на изчисленията – подобряване на алгоритмите, замяна на един вид ресурси с друг (напр. кеширане) и др.
- Намаляване на режийните (overhead) – не-извършване на всякакви изчисления, които не са свързани конкретно с конкретното събитие (което веднага изключва употребата на посредници)
- Промяна на периода – при периодични събития, колкото по-рядко идват, толкова по-малки са изискванията към ресурсите.
- Промяна на тактовата честота – ако върху периода, през който идват събитията нямаме контрол, тогава можем да пропускаме някои от тях (естествено, с цената на загубата им)
- Ограничаване на времето за изпълнение – напр. при итеративни алгоритми.
- Опашка с краен размер – заявките, които не могат да се обработят веднага, се поставят в опашка; когато се освободи ресурс, се обработва следващата заявка; когато се напълни опашката, заявките се отказват.

Тактики за производителност – управление на ресурсите

- ***Паралелна обработка*** – ако заявките могат да се обработват паралелно, това може да доведе до оптимизация на времето, което системата прекарва в състояние на изчакване;
- ***Излишък на данни/процеси*** – cache, load-balancing, клиентите в c/s и т.н.
- ***Включване на допълнителни ресурси*** – повече (и по-бързи) процесори, памет, диск, мрежа и т.н.

Тактики за производителност – арбитраж на ресурсите

- Когато има *недостиг* на ресурси (т.е. спор за тях), трябва да има *институция*, която да решава (т.е. да извършва арбитраж) кое събитие да се обработи *с предимство*. Това се нарича **scheduling**.
- В scheduling-а се включват два основни аспекта – как се приоритизират събитията и как се предава управлението на избраното високо-приоритетно събитие.

Тактики за производителност – арбитраж на ресурсите

- Някои от основните scheduling алгоритми са:
  - FIFO – всички заявки са равноправни и те се обработват подред;
  - Фиксиран приоритет – на различните заявки се присвоява различен фиксиран приоритет; пристигащите заявки се обработват по реда на техния приоритет. Присвояването става съгласно:
    - Семантичната важност;
    - Изискванията за навременност;
    - Изискванията за честота;
  - Динамичен приоритет:
    - Последователно;
    - На следващото събитие, изискващо навременност;
  - Статичен scheduling – времената за прекъсване и реда за получаване на ресурси е предварително дефиниран.